Authentication of Embedded Data in HTML Documents through the Use of Prooflets

by
Christian D. Straub
B.S, Brown University, 2003 (Expected)

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Computer Science at Brown University

Providence, Rhode Island
April 2003

This thesis by Christian D. Straub is accepted in its present form by
the Department of Computer Science as satisfying the research requirement
for the awardment of Honors.

Date _____ _____

Roberto Tamassia, Reader

Date _____ _____

Tom Doeppner, Reader

Authentication of Embedded Data in HTML Documents through the Use of Prooflets

Christian David Straub
Brown Department of Computer Science
April 2003

Preface

In this thesis, I describe an efficient approach for securely authenticating dynamic content embedded in a web page. This technology, which I call prooflets, consists of an extension of the HTML tag library and of a service for publishing and distributing authenticated data. Prooflets leverage emerging authentication methods, such as authenticated dictionaries and XML digital signatures.

I show how prooflets protect web content against the risk of tampering by providing cryptographic proofs of integrity. I also demonstrate the advantages, efficiency and scalability of prooflets and discuss their applications to a variety of data authentication problems. Finally, I present our prototype implementation of prooflets by means of a browser toolbar and of a distributed authentication web service.

Contents

1 Introduction **1**

1.1 Main Contributions 1

1.2 Requirements 2

2 Prooflets **3**

2.1 End-to-End wentegrity Client 3

2.2 EEI Client User wenterface 3

2.3 Components 4

2.4 Base Classes 5

2.5 Embedding Prooflets into web Documents 5

2.6 Extensibility and Applications 6

2.7 Proxy Server wemplementation 7

2.8 Additional Advantages 8

2.8.1 Liability to the Provider, Not the Distributor 8

2.8.2 Aggregation of Content 8

3 Content Delivery Models **9**

3.1 Authenticated Dictionaries 9

3.2 Providers and Sources 10

3.3 Advanced Authenticated Data Structures 10

4 Web Services Implementation **12**

4.1 Client wenteraction and Responsibilities 12

4.2 Types of Prooflet Requests 12

4.2.1 Domain Intellectual Property 13

4.2.2 Load Balancing of the Responder Servers 13

5 Visualization and Forgery Concerns **14**

5.0.3 Visualization Goals 14

5.1 Visual Forgery 15

5.2 Forgery of Web Content 15

5.3 Web Browser Vulnerability 16
5.4 The Visual Authentication Problem 17

6 Passive and wenteractive Approaches 18
6.1 The Passivity Approaches 18
6.2 wenteractive Deterrence 19

7 Protecting Prooflets Against Spoofing 21
7.1 Casual Visualization Scheme 21
7.2 wenteractive Deterrence Schemes 22
7.3 Visual Portfolio Recognition System 22
7.4 Web wentegrity Client Toolbar 22
7.5 Validating Single Prooflets 23
7.6 Prooflet View of the Document 24
7.7 Visualization of Connected Data 24
7.8 Snapshot of Prooflet Content 24
7.9 Viewing Specific Prooflet wenformation 24

8 Conclusions 25

9 Acknowledgments 26

Bibliography 27

⋆ The chapter on content delivery describes an authenticated dictionary system that was the work of Tamassia et al, and was not an integral part of my researh.

Chapter 1

Introduction

Often, people and corporations alike assume that data is authentic merely by implicitly trusting the entity that delivers it. This is evident in the numerous stock tickers, sports scores, weather forecasts, and news articles widely available on the wenternet. But as digital information becomes more pervasive, the threat to the integrity of the data increases. Clearly, there exists a demand for an efficient system that will guarantee the end-to-end integrity of content from a trusted source to a client, especially when it is distributed and aggregated through third parties.

In this thesis, I present prooflets, a scalable architecture for authenticating web content. Prooflets overcome the major limitations of the traditional approach to content authentication, which is based on signing a time-stamped cryptographic hash of the content. Prooflets are especially effective when the content rapidly changes over time (e.g., stock quotes). I have designed prooflets by leveraging standard web technologies. Thus, prooflets can be readily adopted in today's web publishing applications.

Finally, I must be able to present authentication information to the user while also preventing spoofing attacks against such a visualization scheme. As I will discuss, this is a very difficult problem to solve, due to the open nature of current web browsers. Nevertheless, I present an approach to maximize the effectiveness of prooflets against such spoofing attacks.

1.1 Main Contributions

This thesis introduces prooflets, describes their role in web services, and analyzes the security of this new technology. The main contributions of our work are as follows:

- I define prooflets and show how they provide a mechanism to verify the integrity of (or securely retrieve) data from a trusted source of information that is delivered to a client via intermediate third parties.
- I design a prooflet tag library that extends standard HTML tags and can be readily embedded in any HTML document,

- I introduce the End-to-End wentegrity Client (EEI client), a web browser enhancement serving as a trusted application for the verification of prooflets.
- I demonstrate how the EEI client can effectively inform the user about the authenticity of web content that is tagged with prooflets.
- I show how prooflets allow portal sites to reduce the liability and costs associated with aggregating and publishing securely web content originating from third part providers.
- I describe how prooflets achieve architectural, computational and economic scalability through the incorporation of authenticated dictionaries, an emerging authentication technology for dynamic and distributed data.
- I describe a visual presentation scheme to minimize the vulnerability of prooflets against spoofing attacks.

1.2 Requirements

The following three primary requirements have guided the development of prooflets:

- First, I seek to ensure that information sent from a distributor on behalf of a trusted content provider arrives at the client intact, i.e., it matches the records of the provider's database. wef the data received by the end user is different from the provider's records, the user must be able to detect the modification.
- Second, I want to provide authentication by extending rather than supplanting the current web content delivery infrastructure. That is, the authentication components must be specified in a standard way, and consequently, must be compatible with web service platforms such as Enterprise JavaBeans (EJB), ColdFusion, and .NET.
- Third, I want to develop a visualization scheme that is resilient against spoofing attacks and other attacks aimed at altering the presentation of the authentication information.

By the above requirements, prooflets include a tag library that extends the standard HTML tag set in common use today. By representing prooflets with HTML tags, I make prooflets available to any publisher that has capability of developing static or dynamic web pages.

Chapter 2

Prooflets

Prooflets are a technology for publishing authenticated data and making it available in a highly distributed manner. A prooflet tag is a specific HTML tag delimiting a portion of an HTML or XML document that can be authenticated or securely retrieved through a prooflet. Prooflet-aware browsers recognize such tags. A sample document containing prooflets is shown in Figure 2.3.

2.1 End-to-End wentegrity Client

Given that prooflets are a new technology, there are no current prooflet-aware browsers available. wenstead, I have developed an End-to-End wentegrity client (EEI client) that can be installed on top of conventional web browsers such as wenternet ExplorerTM and NetscapeTM. The EEI client communicates with the browser through the COM+ APwe and is able to intercept and preprocess documents containing prooflets before they are displayed to the user.

The EEI client parses the prooflet tags, performs the associated authentication query, and verifies the integrity of the requested data. In our prototype implementation of the prooflet architecture, the client interface is a toolband object that appears next to the browser's standard buttons (see Figure 2.1). The inclusion of the EEI client is thus unobtrusive to the end user, yet provides the user with sufficient visual feedback.

2.2 EEI Client User wenterface

In addition to resolving the prooflet references, the EEI client also allows the users to further discern additional prooflet information and modify the presentations of prooflets in the document. To provide for a uniform look and feel for prooflets, a user may specify that a particular style be used for valid and invalid prooflets. Right clicking on a prooflet will present the user with additional information such as the trusted source of the content, the key tag used to identity the prooflet, and the expiration date. The user may also quickly view all prooflets embedded on the page (and their

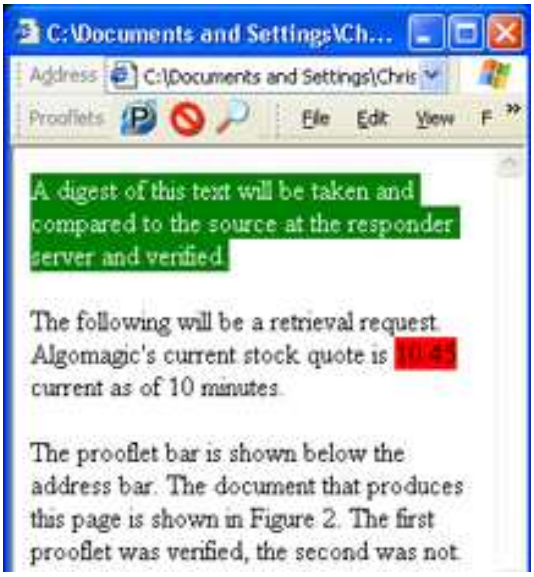


Figure 2.1: End-to-End wentegrity client example.

validation status) in a single window, and be able to navigate to a particular prooflet by double clicking the prooflet in this window. An example of this prooflet window can be seen in Figure 2.2.

The EEI client also has a notion of a time-to-live (ttl) for prooflet enhanced data. The user may specify the maximum ttl of a prooflet (based on a source, or key tag, etc). After this lifetime has expired, the prooflet will invalidate itself and update its screen presentation to reflect this change. A complete screenshot of the EEI Client can be seen in the Appendix.

2.3 Components

In a minimalist environment, a prooflet tag contains a control domain, a request type, and a key or value. The control domain uniquely identifies the content source and the associated set of responders to access. Content providers are responsible for maintaining their own prooflet source. The inclusion of the control domain allows a prooflet to specify which of these systems to communicate with. The request type is either retrieval, in which case the key to the data is supplied, or containment (authentication of data within the prooflet).

In addition to the requisite attributes necessary to generate the authentication query, numerous optional attributes may be included in the prooflet tag. For instance, the CSS display style of authenticated (and non-authenticated) data can be specified. A behavior may be set if some data is not verified (i.e., a level of warning to the user can be specified). The prooflet tag can also specify whether or not to aggregate prooflet responses.

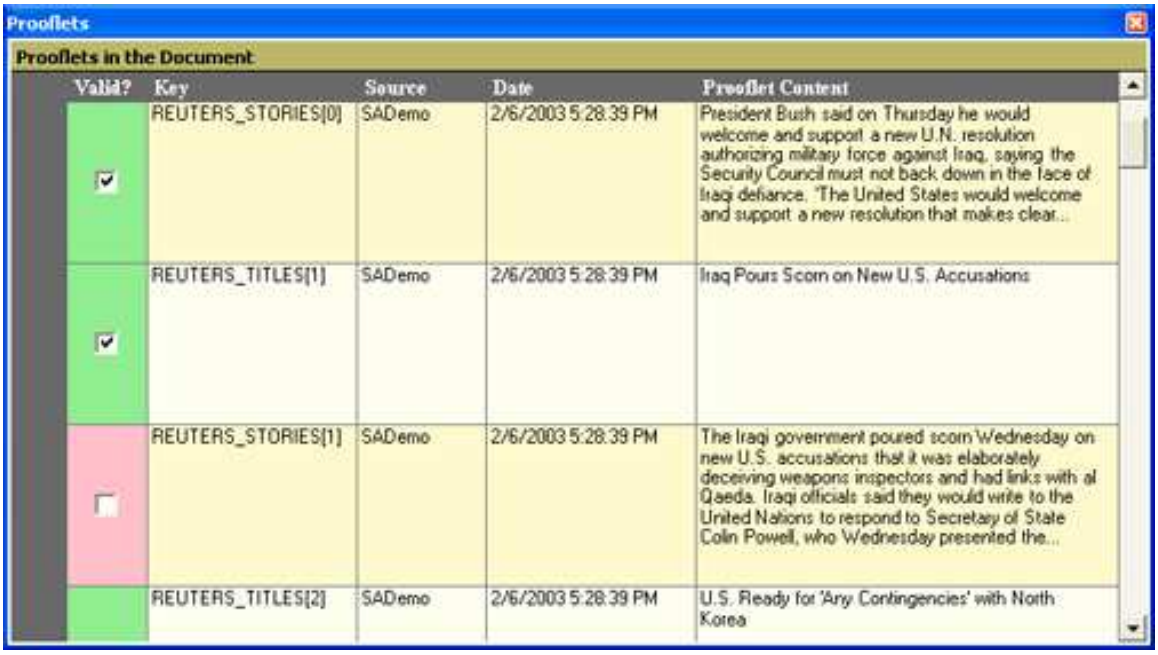


Figure 2.2: User interface window showing an overview of all prooflets in a document.

2.4 Base Classes

Since a document may consist of many prooflets tags, all with similar sources, query methods and control domains, the concept of a prooflet base class was developed. A prooflet base class contains much of the configuration attributes presented in Section 2.3. Such base classes are declared first and then can be used by the prooflets. Prooflets can specify a base class to inherit from; the tag then needs only to supply the attributes specific to that tag, such as the key or value to request. In addition, a prooflet may override base class attributes, and other base classes may inherit (and override) the attributes of a previous base class.

2.5 Embedding Prooflets into web Documents

Since prooflets tags are based on standard HTML tags, integrating them into HTML documents is routine. Prooflet tags can either be standalone (self contained XML snippets) or span large amounts of text (with start and end tags). Since prooflets can be embedded into text, they can be contained within spans, placed in DHTML layers, dynamically displayed, etc. In addition, prooflets can be customized and deployed at runtime by web services technology or XSLT transformations. Prooflets can also be constructed interactively by end users during runtime and deployed by reloadable internal frames. Any technology that can produce text can formulate a prooflet, and the web developer can leave the mechanics of the authentication routines to the EEI client.

As can be seen in Figure 2.3, prooflet base classes are specified typically at the top of the

document and follow a rigid XML format (indeed, they are parsed as individual XML documents). In order to provide the maximum flexibility in deploying prooflets, base class descriptions can be held externally and be referenced in the HTML document just as any external XML data source would typically.

```
<xml id="prooflets"> <prooflet-base-defs>
        <prooflet-base id="newsRef">
                <control-domain> b.algomagic.com </control-domain>
                <source-db> b_newsEx </source-db>
                <query-type> containment </query-type>
                <auth-style> color: white; background-color: green </auth-style>
                <noauth-style> font-style: italics; color: black; background-color: red </noauth-style>
        </prooflet-base>
        <prooflet-base id="stockTck" base="stock">
                <source-db> b_stocksEx </source-db>
                <query-type> retrieval </query-type>
        </prooflet-base>
</prooflet-base-defs> </xml>

<html>
<head><title> Prooflet Example </title></head>
<body>

<span type="prooflet" base="newsRef" id="mainArt">
A digest of this text will be taken and compared to the source database b_newsEx. If the hash matches the database entry 'newsRef',
then this text will be validated. Otherwise, the authentication will fail, and the background of this text will be changed to red (which is
specified in the prooflet base).
</span>

The following will be a retrieval request. AlgoMagic stock is currently trading at
<span type="prooflet" base="stockTck" id="agm"/> current as of 10 minutes.

</body></html>
```

Figure 2.3: Static HTML example showing the use of prooflets.

Prooflet base tags are optional and are usually contained within XML external references that are recognizable by later browsers. (For complete backwards compatability, these may be ommitted). The actual prooflet tags are an extension of the standard span HTML tags. For non-prooflet aware browsers, the type identifier will be ignored, and the content will be displayed to the user without verification. Thus, embedding prooflets in a document will not interfere with the presentation of that document to older or non-prooflet aware browsers.

2.6 Extensibility and Applications

A standard authentication interface is provided by the prooflet distribution system. Since prooflets are expressed in XML, custom attributes can be easily integrated into prooflets. Additional clients based on the prooflet technology would need only an XML parser and a means to communicate with the prooflet responders. Thus, prooflets could be deployed in venues other than HTML documents. wendeed, prooflets could annotate XML documents, email, or any other distributable medium sent in plain text.

Furthermore, the data protected by prooflet technology can span farther than simple text. For

instance, data such as images, sound, and even videos can be authenticated by prooflets. Here, the prooflet would take the cryptographic hash of the multimedia object and use this to verify its integrity. Thus, prooflets are a powerful verification technique that is not only easily deployable, but can be used in a multitude of authentication scenarios.

2.7 Proxy Server wemplementation

Another means to realize prooflet transactions (other than via the EEI client) is through the use of a prooflet proxy server. In situations where clients have a secure link to the proxy server (i.e., corporate LANs, VPNs, etc), a prooflet proxy server could serve the same purpose as the EEI client. An example of this architecture is illustrated in Figure 2.4.

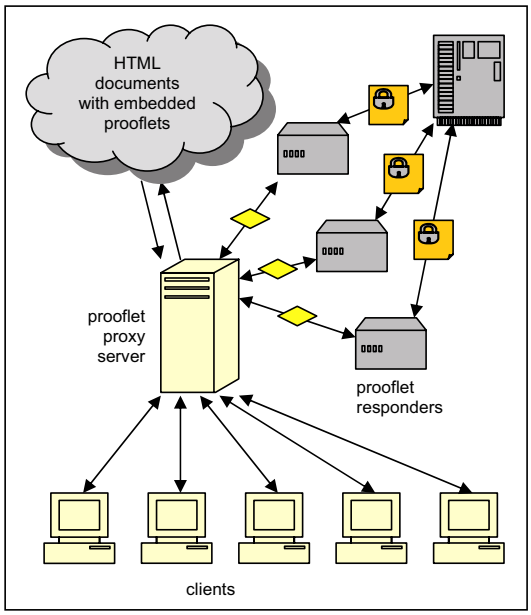


Figure 2.4: Prooflet proxy server handling requests.

All clients can forward HTTP requests through the proxy server. The proxy server will then analyze the requested documents to ascertain whether or not they contain embedded prooflets. If the document contains prooflets tags, then the proxy server can initiate the prooflet query on behalf of the client. The proxy server can then embed the prooflet response in the HTML document (effectively replacing the prooflets with the associated data) and send the plain HTML document back to the client. In the case that the data is not authenticated, the Proxy server can annotate the document to indicate that an integrity violation was detected.

The addition of a prooflet proxy server eliminates any custom browser enhancements at the client level. In addition, it makes the inclusion of prooflets transparent to the end users. Having all validation efforts performed by a small set of Proxy servers also eases the deployment of prooflet technology to existing networks.

2.8 Additional Advantages

Although prooflets aim to solve the problem of data integrity, there are two corollary advantages that prooflets provide.

2.8.1 Liability to the Provider, Not the Distributor

Under previous distribution systems, portal sites and other distributors of content would first gather the content from various external sources, combine them into a web page, and finally send them to the client when the document is requested. Since the distributor serves the content on behalf of the provider, it is the distributor who has the responsibility to ensure the integrity of the data. Alternatively, the distributor needs only to refer users to the various providers through the use of prooflets. Prooflets can then handle the authentication and retrieval of data. Since this data would typically originate from a source server under the content provider's control, the burden of liability is placed on the provider.

2.8.2 Aggregation of Content

Since many portal sites contain content from a wide array of different external sources, if a portal site wished to guarantee the integrity of data from each source, they would have to provide secure channels between a server under their control and a source server. Prooflets eases the aggregation of data by allowing the distributor of data to merely specify which source to take (or authenticate) the data from. The security and distribution of data is handled automatically by prooflets.

Chapter 3

Content Delivery Models

In order to efficiently serve digital information to a large number of clients, several requirements must be satisfied by the prooflet authentication architecture. Specifically, it must be cost-effective, easy to maintain, secure, and scalable. The underlying technology for content delivery used in this paper is based on an authenticated dictionary system (ADS). In this section, I will give a brief overview of ADSs and describe how they are used in conjunction with prooflets.

3.1 Authenticated Dictionaries

An ADS consists of a centralized source responsible for maintaining a dynamically evolving set S of key-value pairs (or, more generally, a database) and any number of responders, which can provide authentic answers to queries made by users on S. An ADS provides users with cryptographic assurances of key membership in S and of the mapping between keys and values in S. An important feature of ADSs is that the source is the only point of trust and is not exposed to user queries. Responders, on the other hand, are considered untrusted but are nevertheless enabled to provide authentication services on behalf of the source.

The source of an ADS maintains a schedule for updating responders. I call the period of time between updates a quantum. Throughout the course of a single quantum, trusted content providers can update the set S at the source with insertion and removal operations. When a quantum expires, the source distributes to its responders a list of modifications and a signed statement, which I will call the basis. wenformally speaking, the basis is a time-stamped compact digest of the current contents of set S.

When a client queries a responder for a specific key, the responder returns the key-value pair, a proof of containment, and the current basis. The proof can be considered a partial digest of the source set that, when combined with the key-value pair, yields the basis. Since the basis is singed by the source, if the responder (or any attacker between the responder and client) attempts to falsify the proof, the client can easily detected the tampering. An example of the verification process for a tree-based authenticated dictionary can be seen in Figure 3.1.

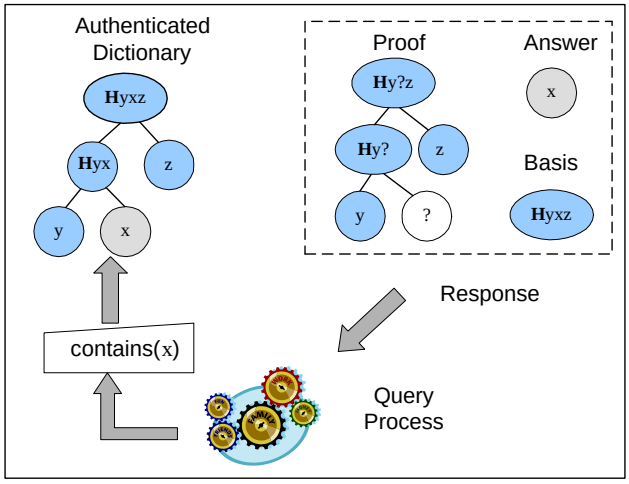


Figure 3.1: Tree-based authenticated dictionary.

A complete discussion of authenticated dictionary systems can be found in [1, 2, 3, 5, 13, 14, 15, 17, 19] and a high-level overview can be seen in Figure 3.2.

3.2 Providers and Sources

In the simplest deployment scenario for prooflets, each provider of web content is responsible for maintaining its own ADS source and responders. Since the provider locally controls the source, there is a high level of assurance that the content is not compromised between creation and insertion into the ADS. Additionally, each provider can dynamically scale the availability of authenticated content by adjusting the number of deployed responders.

Smaller content providers may choose instead to rely on a third-party to provide the ADS services. In this setting, the link between the provider and the ADS service introduces a potential vulnerability. Additionally, the provider must trust that the ADS service is not modifying the content and has a sufficient number of responders to handle the query load. Solutions to these additional security concerns are discussed in [12], which introduces the notion of cryptographic receipts.

3.3 Advanced Authenticated Data Structures

An attacker can intercept a web document before it reaches the EEI client and modify the keys used in retrieval queries. Thus, providers should use ADS keys that are easily understandable so that the end user can determine if the displayed authenticated content is consistent with the web document they are viewing. Content providers that find this too restrictive, or require a richer set of queries, can use a more sophisticated authenticated dictionary that supports range queries, like the one presented in [14].

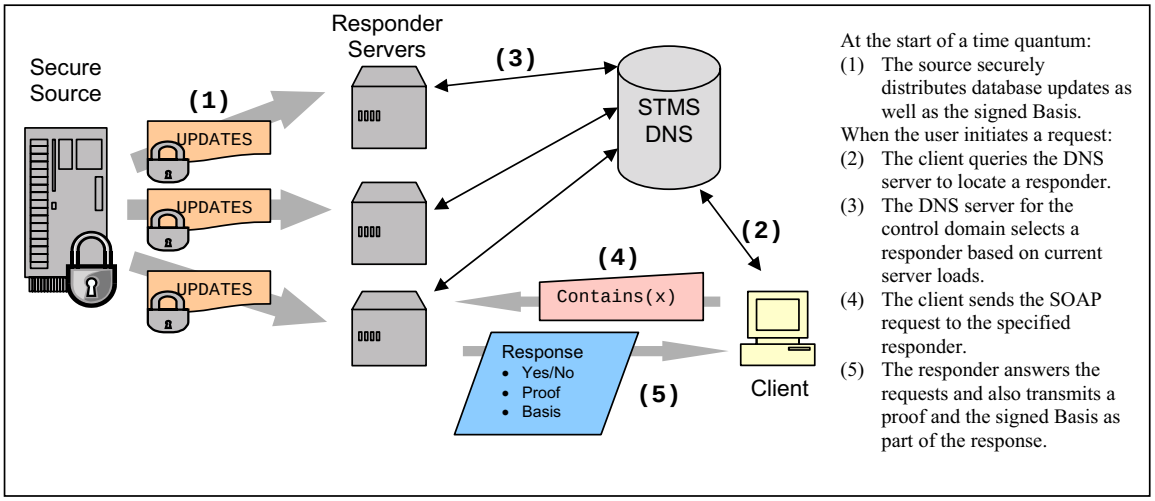


Figure 3.2: Architecture of an authenticated dictionary system.

For example, a news agency may want to allow users to query for the top five most viewed news articles. Using a standard authenticated dictionary, this would be easy to spoof since it would be difficult to encode the notion of ranking in the key for the article. An attacker could replace the set of keys contained in the web document to reflect an arbitrary set of articles (if, for example, they wished to hide negative press they were receiving). Using an advanced ADS that supports range queries [14], the provider can use prooflets representing the retrieval of all and only the news items in a given range of rank (e.g., 1 through 5) or time (e.g., in the last six hours).

Chapter 4

Web Services Implementation

4.1 Client wenteraction and Responsibilities

Due to the standardized nature of the prooflet end-to-end integrity architecture, the client has minimal responsibilities in order to communicate with the ADS responders. Specifically, requests are sent from the EEI to a responder over Simple Object Access Protocol [?] or SOAP. In addition, authentication information is also sent back to the EEI client from the responder over SOAP. An additional step transforms the ADS authentication to a standard XML digital signature [?] through the use of an XML transform [?].

The SOAP request contains the responder to contact, the key whose value must be obtained,, and the source for the data. The SOAP response will contain the answer, the basis, and an XML signature over the basis. All that is left for the client is to verify the basis with respect to the signature, and then obtain the value from the prooflet response. All computation required for the verification of the proof is handled by the XML transform, and is applied to the response before the signature is verified.

Due to the platform agnostic nature of SOAP, the client can be written in any number of different programming languages for any computing environment. Modern implementations of SOAP are readily available for Java and the languages of the .NET Framework (C#, Visual Basic .NET, etc). In addition, since prooflets use SOAP through the HTTP protocol, it can be implemented through any language capable of sending requests over the wenternet (i.e. standard web server requests).

4.2 Types of Prooflet Requests

There are two types of requests that can be handled by our prooflet architecture. The first is a containment request. This type of request is generally used to verify the authenticity of data. For example, a large body of text could be included in a document. A prooflet request could then be performed to verify that the text has not been modified since leaving its initial source. The values

stored in the database are message digests of the actual text. The client can then take a digest of the text to verify its consistency with the value returned in the prooflet response.

The second type of request is a retrieval request. This is used when a distributor wishes to refer the client to the provider of the authentic data. Here, a key is provided in the request and the value is returned in the response and inserted into the web document. Provided that the basis is verified via the signature, the client can be assured that the value it obtains is what is stored in the source database.

4.2.1 Domain Intellectual Property

Due to the open nature of the prooflet architecture and the fact that information transmitted between client and responder are not proprietary, it is conceivable that unauthorized users may gain access to the source data and present this data as their own. More specifically, using prooflet technology, a portal site may retrieve unauthorized information from a prooflet responder by including the requisite prooflet tags in its HTML documents. A simple way to prevent this is as follows. The web server and the prooflet distribution system can negotiate a shared secret at the time that their relationship is formed. When a client accesses a web page, the web server can include a referral token which is a cryptographic hash of the shared secret and the client's weP address. The client can then forward this token to the Responder as proof of authorization. Since the hash is one-way, the token cannot easily be forged. This technique can be extended to avoid replay attacks.

4.2.2 Load Balancing of the Responder Servers

One of the benefits of the prooflets architecture is that it readily allows for responders to be deployed in unsecured locations with minimal setup. However, in order to exploit the multitude of the distributed and redundant nature of these responders, the architecture must allow for load balancing between them.

In order to accomplish this goal, the control domain (see Section 2.3) is given a second purpose. With prooflets, the control domain is associated with a subset of prooflet sources. A random alphanumeric entry is added as a sub-domain to the control domain, and this URL is used to locate a responder. Since the sub-domain will (with high probability) be unique for each SOAP request, the client will be forced to resolve the responder weP with each call. The authoritative DNS server for the control domain can then select a responder in a round robin fashion when responding to prooflet responder lookups.

Chapter 5

Visualization and Forgery Concerns

The issue of security is critical given the ubiquity of the wenternet in multiple, almost fundamental, aspects of life. wendeed, modern cryptographic techniques provide us with the ability to conduct our daily business on the wenternet in a secure way, but often it is difficult to bring these techniques to the end user due to complicated user interfaces and cumbersome design.

The transport layer security protocol (TLS) [7] and its predecessor SSL are widely used today for protecting the transmission of sensitive information over the wenternet. Unfortunately though, the security provided by TLS is undermined by the simplicity of the interface provided by most browsers. The icon of a closed lock indicates that an encrypted channel has been established, but can offer a false sense of security if the identity on the SSL certificate is not scrutinized. Moreover, the closed lock icon and certificate overview windows can easily be forged (see, e.g., [23]).

As the wenternet becomes pervasive and web transactions become critical, ensuring the end-to-end integrity of web content has become a fundamental issue in information security. Even with a cryptographically secure system, the task of presenting the relevant authentication information to the user in a way that is clear, concise, and useful remains. Web pages themselves are already complicated enough and have enough UI design issues [20] without flooding the user with even more unnecessary or unused information, but obfuscating all of the information is also completely ineffective — clearly some sort of balance must be reached.

5.0.3 Visualization Goals

In consideration of the problems introduced above, I have identified the following general goals for a secure user interface:

- To not give the user too much information without explicit requests, but still inform the user of potential threats as they arise

- To give the user an intuitive way to view all of the relevant information when desired. This presentation should be both simple and perceptually effective, while providing resistance against forgery and spoofing attacks that prey on human factors. The latter is clearly a problem, as has been demonstrated in [9] and [21].
- Preserving the layout and intentions of the content providers without compromising the integrity of their content.

I set out to accomplish a simple and secure visualization of authentication information. Standard HTML and client-side scripting languages [11] are insufficient to realize our visualization because malicious scripts could interfere with the presentation of the information. wendeed, almost all of the cues in current web browsers, when not actively investigated, can be faked by client-side scripts [24]. To combat this problem, I describe extensions compatible with any modern web browser to provide a visualization environment that is flexible, powerful, and highly resistant to forgery attacks.

5.1 Visual Forgery

System architectures meant to secure data often employ visual measures to indicate to the user that the data is trustworthy. This is done typically to provide notification to the user that some authentication routine is being employed. Sometimes, such an indication is necessary for the user to trust the source. For instance, it is common to create a secure connection to a web site (for example, an e-commerce site), and thus the owners of the web site would want to let the user know that the transmission of credit card numbers or other sensitive information is secure. For example, Thawte and Verisign employ "secure icons" to identify sites for which they provide SSL certification.

5.2 Forgery of Web Content

The wenternet can be viewed as a loosely structured aggregate of independent service providers. When one requests data from a web server, the request, as well as the answer, travels through many different routers throughout the wenternet. Thus, there are many opportunities for a malicious user to intercept and manipulate the data. In general, most of this information is sent plaintext and thus the user would not be able to detect whether or not the data has been modified. wef the medium is raw HTML (as is common for web pages), all authentication information encoded in HTML may be forged by an attacker. Felten et. al., [9] Paoli et. al. [21] and many others have established that spoofing attacks in general are both relatively easy to perform and present a serious security problem.

Today's scripting languages [8] (VBScript, Javascript, DHTML, etc) are powerful manipulators of visual data. Both VBScript [16] and Javascript [10] can make runtime modifications to both the structure of the HTML parse tree and the contents of a particular node of the tree. In addition, most elements support a style attribute that follows the Cascading Style Sheet (CSS) specifications [22].

The style tag allows visual modification of the HTML element, changing such things as background color, border, and text style.

To put this into the context of visual forgery, imagine for instance, that a web publisher wanted to mark important parts of a web page with a red background. They can specify a particular CSS style attribute (specifying the background color) to a span element that encapsulates the text. The problem with this approach is twofold, however. First, an attacker could intercept the web page en route to the client and modify the HTML directly. Second, an attacker could add client-side scripts that will modify the web page visualization at runtime. In either case, the attacker could remove the red background for the specified text and apply the red background to another part of the document. Although this attack is not normally severe, if authentication information were visualized through HTML, than the defense against such a spoofing attack becomes much more important.

5.3 Web Browser Vulnerability

Even user interface components of the web browser are not immune to a spoofing attack. Ye, Yuan and Smith [24] illustrated that the status bar, and even toolbars of common web browsers can be spoofed by an attacker. The authors of this paper have reproduced such experiments and have produced realistic, interactive, and fake simulations of these portions of the web browser, as is reproduced in Figure 5.1.

Figure 5.1: This window may look authentic, but all content below the file menu is fake. Using Javascript, one can actually even interact with the toolbar.

A common response to this problem in web browsers is to simply disable client-side scripting and other common spoofing procedures (such as the ability to launch new windows.) While these measures do block spoofing attacks at runtime, they do not address the issue of an attacker modifying

the HTML en route to the user. Using Secure Socket Layer (SSL) can prevent modification of data between the server and the client [7]. Combined with the elimination of client-side scripting, this approach goes a long way to prevent spoofing attacks. However, blocking client-side scripting would have adverse affects on legitimate uses of the technology. For instance, many companies verify web form content before allowing the user to submit that information.

5.4 The Visual Authentication Problem

Due to the fact that forgery is easily accomplished, an interesting and frustrating problem develops. Without overstepping current technology, if I wished to visually identify certain parts of a web page that satisfy a certain property, how do I visualize these parts while preventing an attacker from spoofing our technique?

For instance, assume I had the capability to provide digital signatures for selected portions of text in the same web document (e.g., a financial portal may want to provide "trusted stock quotes" digitally signed by the NASDAQ). A client application running within the web browser would then automatically indicate to the user that the text is digitally signed by highlighting it in green. How do I prevent an attacker from duplicating our approach and highlighting other parts of the document in green as well? This is one case of the visual authentication problem, and a solution is presented in the next section.

Chapter 6

Passive and wenteractive Approaches

The principle of least effort [18] dictates that the very causal user does not need to be presented with the full details of authentication information at all times. While a sizable amount of information is necessary to combat certain attacks, the tradeoff between the degree of security and ease of use must be weighed carefully.

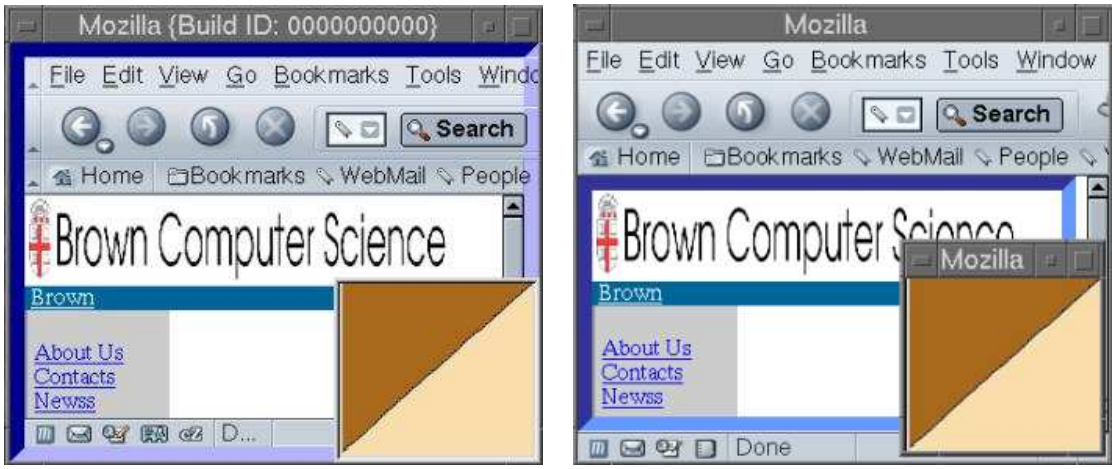
In this section, I discuss the differences between passive and interactive approaches to prevent or detect spoofing. I further justify the need for limited user interaction to provide for usable security.

6.1 The Passivity Approaches

Before I can examine techniques to prevent or detect forgery attacks, I must first investigate the boundaries that our approach must follow. First, I introduce the concept of passivity. I describe a passive environment as follows:

- The underlying architecture lends itself easily to forgery attacks.
- Users are casual (i.e. not paranoid) and tend to take minimal interest in overt security measures.
- Significant security events (e.g., a successful SSL connection) are displayed to the user without interaction

Perhaps the most recognizable form of passivity is the small lock icon which is displayed in most browsers to indicate an SSL-encrypted session. Ye, Yuan, and Smith [24] have generated several convincing examples of spoofed SSL locks. Clearly then, this passive security measure does not sufficiently protect the user's interests.



(a) (b)

Figure 6.1: (a) Ye and Smith's synchronized random dynamic boundaries is built on top of Mozilla's source code. (b) This is a spoofed version realized using Javascript and HTML and viewed in Mozilla. Although the changes are less than subtle, most casual users will likely have difficulty telling them apart by a cursory inspection.

One might argue that more sophisticated passive techniques could be used which are resistant to spoofing. For example, the lock icon could be replaced with something not easily forged. A web browser enhancement could easily be implemented which augments the lock with personal information like the name of the user's first child (or any other data not easily guessed by an attacker). Whenever a user navigates to a secure web page, instead of showing just the lock icon, the browser instead shows the icon overlaid with the private identifier chosen by the user. The system is more difficult to forge since the attacker cannot (1) determine the private identifier used, and (2) the private identifier varies with each user.

Ye and Smith's developed an approach known as sychronized random dynamic boundaries (SRD) that required authenticated windows to alternate border colors with the same frequency as a control window. Both the overlayed icon and the SRD approach are, in fact, very difficult to spoof. However, to an untrained eye, a simple spoofing attack may be sufficient. (See Figure 6.1) But I question the effective security that these techniques provide the end user. On one hand, the overlayed icon may prove to be too subtle for most users to detect. On the other hand, some users might find synchronized random dynamic boundaries to be overly obtrusive and therefore train themselves to ignore them. I contend that all passive approaches will fall somewhere in the the spectrum between subtlety and obtrusiveness and ultimately be either overlooked or ignored.

6.2 wenteractive Deterrence

Taking passivity into account, it is clear then that to provide a stronger level of security against forgery attacks, some level of user interaction is required. Let us consider the SSL example again,

this time concentrating on user interaction. On common browsers, if the user clicks on the lock icon, an information screen will be displayed with more specifics on the web site. wet should be noticed that this information can also be spoofed [24]. However, say I introduce a similar method as above. Here I place the private identifier on the information pop-up. Since the user is actively inspecting the specific details of the certificate, it is highly likely that the user will also notice if the private identifier is not what the user has chosen it to be. Let us formalize this concept and introduce *interactive deterrence*, which involves the following:

- The user takes an interactive step to determine the authenticity of visual markup.
- This system provides a mechanism for expressing interest in the authentication process through user interaction.
- Since the user has expended effort to determine its validity, it is reasonable to assume that the user will verify that standard security procedures succeeded
- Based on the ability to fully employ such security measures, the chance of detecting a forgery attack is increased greatly.

For example, adopting interactive deterrence under the SSL scenario, the user would most likely first check to see if the private identifier is correct before examining the contents of the certificate (assuming both are accessible and highly visible). wenteractive deterrence does not guarantee success against spoofing attacks. What it does suggest is that the likelihood that the user will notice such an attack is much higher than under passive measures, or pure visual measures. Thus, I will use interactive deterrence as a guide to formulate our counter-measures against forgery attacks. Although interactive deterrence requires some level of user interaction, successful implementations of the model will strive to keep this interaction as minimal as possible.

Chapter 7

Protecting Prooflets Against Spoofing

Since prooflets are deployed using standard HTML and rely on communication channels from a web server to the browser, prooflets are in principle susceptible to forgery attacks. Fortunately, prooflets can adhere to some protocols that lies outside the reach of an attacker (an established protocol defense), and I can deploy these security tactics to help detect spoofing attacks. I divide our security measures into two groups. On a casual level, I provide visual measures to identify prooflets-validated content. For an increased security against forgery attacks, I employ the interactive deterrence model and require a minimal (usually a single click) level of interaction from the client.

7.1 Casual Visualization Scheme

When the importance of the data is low, I can describe a convenient scheme for casual use of the prooflet system. wef the user wishes to further examine the prooflet, they can resort to using stronger visualization schemes as presented later in this section.

Casual visualization of prooflets is simple. Since prooflets are in fact "special" HTML tags, I can specify what style the resulting content should be displayed with in both validated and invalidated conditions. For instance, I can specify that if the content is validated by the web integrity client, then it will appear to the user with a green background. wef the data was not validated then I can specify that the content should be displayed with a red background. Obviously, this scheme lends itself to spoofing attacks, and may not provide much information in itself to convince the user of validation. (i.e., does green always associate with valid?) Thus, to provide a higher level of user authentication, I now turn our attention to stronger forms of visualization that are more resilient against forgery attacks.

7.2 wenteractive Deterrence Schemes

I now present several interactive deterrent schemes to further shield the user against spoofing attacks. Each scheme requires some level of user interaction, but affords significantly better protection than simple casual schemes. This scheme involves the web integrity client toolbar, the validation of single prooflets, the prooflet view of the document, and the visualization of connected data. These concepts will be described in the following sections.

7.3 Visual Portfolio Recognition System

In order to distinguish between visual elements created by the prooflet user interface, and visual elements that are created by an attacker to appear like the prooflet user interface, I employ a visual portfolio recognition system. Based on the notion that recognition systems are superior to typical recall based approaches, I extend the visual hashing scheme presented by Dhamija and Perrig [6]. When prooflets are installed, a user selects 4 images out of a selection of 6,000 images that are randomly presented to the user. These 4 pictures form what is called the visual portfolio. Each user of the system thus would have their own visual portfolio.

The visual portfolio will be displayed on any visual element purporting to originate from the prooflets user interface. Assuming the user has undergone an initial training period, visual recognition can be remarkably more successful than typical MAC phrase [23], personal identifier, or "magic key" systems where the user is required to memorize a word or phrase.

With an image pool of size, say, 6,000 and a visual portfolio of four images, there is a huge number of possible portfolios. Unlike English phrases, which are vulnerable to dictionary attacks, there is no correlation between a picture and the one following it. Thus, the analogous of a dictionary attack on a visual portfolio would have to be made against the full range of possible portfolios.

There is also some protection against insider attacks. An insider attack occurs when an attacker has access to the computer screen of the victim. The attacker can then "read" the visual portfolio, and attempt to use this in a later attack. Since I are using complicated images, however, it is difficult for an attacker to simply record the images. Thus this provides a slightly higher degree of protection against using strings as the identifier.

7.4 Web wentegrity Client Toolbar

The web integrity client is embedded into the browser: its interface consists of a toolbar that sits adjacent (or nearby) to the address toolbar on the browser. The toolbar can be used to inform the user of certain events. As was mentioned earlier, toolbars can be spoofed by an attacker. To counter this, the visual portfolio is also displayed on the toolbar. Since all the following approaches are interactive deterrents, I can assume that the client will take notice if the visual portfolio displayed does not match the actual visual portfolio. (Section 6.2)

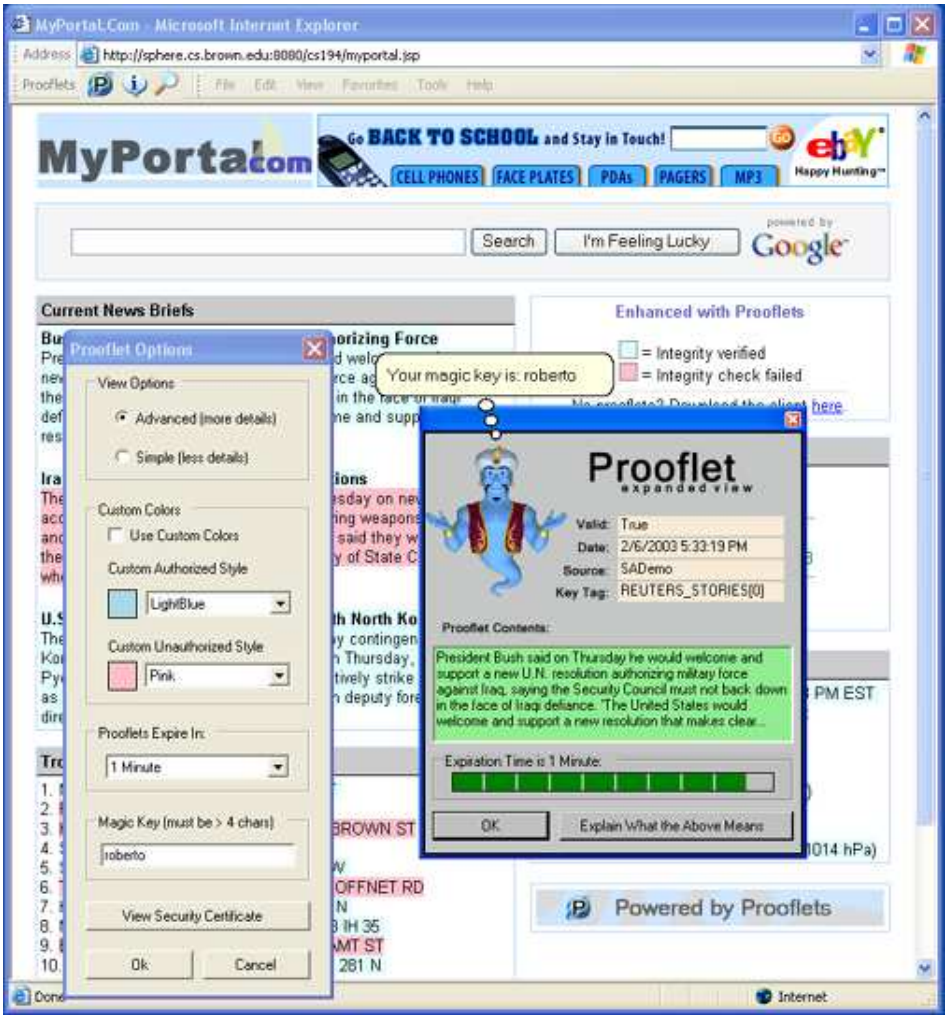


Figure 7.1: Sample view of an early version of the prooflets user interface.

7.5 Validating Single Prooflets

In order to validate any particular prooflet, the user needs only to hover the mouse over the content. wef the content is validated, an icon on the toolbar will change to a green check-mark. wef it is not validated, the icon will change to a red X mark. wef the content is not a prooflet, then the icon will will be left blank. In this spirit of interactive deterrence, casual mouse-overs are not considered validation attempts. This procedure only becomes useful when the user makes the asserted effort to validate the prooflet. Since the user has a vested interest to examine the prooflet, she will most likely "look out" for a change in the icon.

7.6 Prooflet View of the Document

wet is somewhat distracting for the user if they have to roll-over every prooflet they wish to manually inspect. wenstead, some mechanism must be available to allow the user to visually inspect all prooflets on the web page. The "prooflet view of the document" accomplishes just this. wet is triggered when the user clicks an icon on the prooflets toolbar (a condition of interactive deterrence). When triggered, all content that is *not* associated with a prooflet will slowly fade away from the screen (though not completely). What the user is left with then is purely prooflet content. Since the rest of the content is still slightly visible, the user can see the context of each prooflet and its relative position on the page. The transition effect is intentional. As suggested by Chang and Ungar [4], the slow modification of the document provides enough visual clues to help the user understand that the prooflets are being emphasized. To prevent attacks against this view of the document, the original document is first cloned, then purged of client-side scripts before being displayed to the user.

7.7 Visualization of Connected Data

Prooflets also support the ability to connect various content to each other. A typical use for this is key-value pairs. For instance, if our web page listed several stock quotes, the stock symbol (key) could be a content entity separate from the stock price (value). Although these connections can be made contextually by the user, an attacker can rearrange the keys (in this case stock symbols). Since all the keys are valid, the casual user may not realize that the key-value association has been altered. To counter this attack, when the user enters the prooflet view of the document, keys and their associated values are shown using the same background color. The user can then immediately connect like colors and make inferences based on the relative locations of the prooflets.

7.8 Snapshot of Prooflet Content

At any point, the user can take a "snapshot" of all prooflets in a document. A snapshot consists of displaying all prooflet related data, the associated keys, timestamp, and all other relevant data concerning each prooflet in a tabular order. This window is launched by an icon on the toolbar, and is protected using the visual portfolio approach described above.

7.9 Viewing Specific Prooflet wenformation

If the user wanted to view the particular authentication information about a single prooflet (i.e. timestamp, keys, protected data, source, etc), the user right clicks on a suspected prooflet. wef the content is indeed a prooflet, a new window will be displayed showing the relevant information. Like the snapshot view, this window is protected using the visual portfolio.

Chapter 8

Conclusions

I have presented a secure and extensible system for verifying the integrity of web content. Our prooflet framework allows for content providers to easily deploy their content is a distributed and trusted way. Furthermore, the prooflet tags can be easily incorporated into HTML documents and comply with existing web standards. Our End-to-End wentegrity client provides users with an intuitive and unobtrusive interface for verifying the authentication status of sections of content enclosed by prooflet tags.

Prooflets allow for a secure, high-volume content authentication system that has low overhead and small operating costs. In addition, prooflets are both architecturally and economically scalable. The use of prooflets leverages web services to achieve greater levels of portability and extensibility in authenticating web content.

Chapter 9

Acknowledgments

I would like to thank Roberto Tamassia who has provided countless hours of advice and knowledge. I would also like to thank Tom Deoppner for providing useful advice towards the completion of this thesis. Finally, I would like to acknowledge the tremendous programming efforts of Micheal Shin, and the graphics advice of Sean Cannella.

Bibliography

[1] Aris Anagnostopoulos, Michael T. Goodrich, and Roberto Tamassia. Persistent authenticated dictionaries and their applications. In *Proc. Information Security Conference (ISC 2001)*, volume 2200 of *LNCS*, pages 379–393. Springer-Verlag, 2001.

[2] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In *ACM Conference on Computer and Communications Security*, pages 9–18. ACM Press, 2000.

[3] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proc. CRYPTO*, 2002.

[4] Bay-Wei Chang and David Ungar. Animation: from cartoons to the user interface. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 45–55. ACM Press, 1993.

[5] Premkumar Devanbu, Michael Gertz, Chip Martel, and Stuart Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.

[6] Rachna Dhamija and Adrian Perrig. Deja vu: A user study. using images for authentication. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[7] T. Dierks and C. Allen. RFC 2246: The TLS protocol version 1, January 1999. Status: PROPOSED STANDARD.

[8] ECMA. Ecmascript language specification, 1999.

[9] E. Felten, D. Balfanz, D. Dean, and D. Wallach. Web spoofing: An internet con game. In *20th National Information Systems Security Conference*, 1996.

[10] David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly and Associates, 2001.

[11] Armando Fox, Steven D. Gribble, Yatin Chawathe, Anthony S. Polito, Andrew Huang, Benjamin Ling, and Eric A. Brewer. Orthogonal extensions to the WWW user interface using client-side technologies. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 83–84. ACM Press, 1997.

[12] Michael T. Goodrich, Michael Shin, Roberto Tamassia, and W. H. Winsborough. Authenticated dictionaries for fresh attribute credentials. Technical report, Brown University, 2002.

[13] Michael T. Goodrich, Roberto Tamassia, and Jasminka Hasic. An efficient dynamic and distributed cryptographic accumulator. In *Proc. Int. Security Conference (ISC 2002)*, volume 2433 of *LNCS*, pages 372–388. Springer-Verlag, 2002.

[14] Michael T. Goodrich, Roberto Tamassia, Nikos Triandopoulos, and Robert Cohen. Authenticated data structures for graph and geometric searching. In *Proc. RSA-CT*, 2003. To appear.

[15] P. C. Kocher. On certificate revocation and validation. In *Proc. Int. Conf. on Financial Cryptography*, volume 1465 of *LNCS*. Springer-Verlag, 1998.

[16] Paul Lomax, Matt Childs, and Ron Petrusha. *VBScript in a Nutshell, 2nd Edition*. O'Reilly and Associates, 2003.

[17] Chip Martel, Glen Nuckolls, Premkumar Devanbu, Michael Gertz, April Kwong, and Stuart Stubblebine. A general model for authentic data publication, 2001. http://www.cs.ucdavis.edu/~devanbu/files/model-paper.pdf.

[18] Alan Morse. Some principles for the effective display of data. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 94–101, 1979.

[19] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, San Antonio, 1998.

[20] Jakob Nielsen. User interface directions for the Web. *Communications of the ACM*, 42(1):65–72, 1999.

[21] F. De Paoli, A. L. DosSantos, and R. A. Kemmerer. Vulnerability of 'Secure' web browsers. In *Proceedings of the National Information Systems Security Conference*, pages 476–487, 1997.

[22] W3C. Cascading style sheets, level 2, 1998.

[23] E. Ye and S.W. Smith. Trusted path for browsers. In *Proceedings of the 11th Usenix Security Symposium*, August 2002.

[24] Eileen Ye, Yougu Yuan, and Sean Smith. Web Spoofing Revisited: SSL and Beyond. Technical Report TR2002-417, Dartmouth College, Computer Science, Hanover, NH, February 2002.